PCI Vault Technical Specification

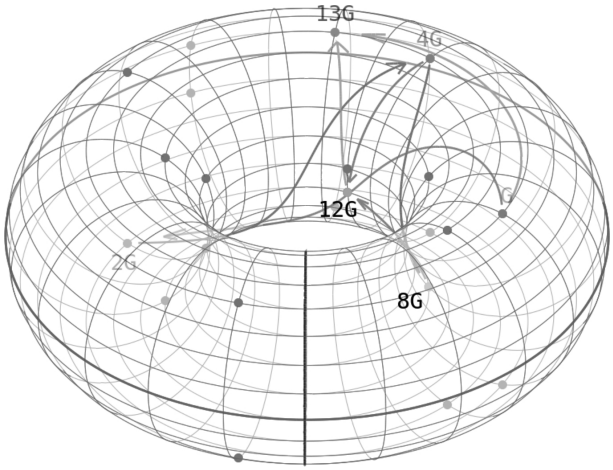
A Zero-Knowledge (ZK), Serverless Architecture for
Secure Payment Data Life-cycle Management with Reduced Liability

JJ van Wyk
BEng (Computer), Cryptography

I Todorov
BSc (IT), BSc (Hons), Computer Science

March 25, 2026

Website: https://pcivault.io
Documentation: https://docs.pcivault.io



Abstract

This technical specification outlines the architectural design and cryptographic methodologies employed by PCI Vault, a zero-knowledge, serverless solution designed to address the complexities of Payment Card Industry (PCI) compliance.

We explore the challenges organizations face with respect to data security standards and present a comprehensive solution to securely store, tokenize, and transmit sensitive card and account data.

This document provides an in-depth analysis of the selected algorithms, including x25519 for key exchange, BLAKE2s for hashing and the Luhn algorithm for validation, along with the benefits of adopting ubiquitous data formats like JSON and security standards such as TLS and Subresource Integrity (SRI).

Contents

1 Introduction **3**
- 1.1 Definitions . . . 3
- 1.2 The Compliance Challenge . . . 3
- 1.3 The PCI Vault Solution . . . 3

2 Infrastructure and Security Architecture **3**
- 2.1 Serverless Environment . . . 3
- 2.2 Transport Layer Security (TLS) . . . 3

3 Key Management and Cryptography **4**
- 3.1 Client-Controlled Security . . . 4
- 3.2 x25519 for Key Exchange and Encryption . . . 4
  - 3.2.1 Benefits of x25519 . . . 4

4 Data Vault and Tokenization Strategy **4**
- 4.1 JSON Data Structure . . . 4
  - 4.1.1 Benefits of JSON . . . 5
- 4.2 Tokenization Algorithms . . . 5
  - 4.2.1 BLAKE2s Hashing . . . 5
  - 4.2.2 Token Formats . . . 5

5 Secure Data Integration **6**
- 5.1 Third-Party Forwarding . . . 6
- 5.2 Capture Endpoints and Webhooks . . . 6
- 5.3 Hosted HTML Forms with SRI . . . 6
  - 5.3.1 Subresource Integrity (SRI) . . . 7

6 Rule Engine **7**

7 Conclusion **7**

1 Introduction

1.1 Definitions

To ensure clarity throughout this document, the following definitions are established:

- **Client**: An organization that uses the PCI Vault system to securely store sensitive financial data.
- **API Consumer**: A server or automated system operated by the Client that interfaces programmatically with the PCI Vault API.
- **End User**: The individual whose sensitive card or account details are being captured and protected.

1.2 The Compliance Challenge

The handling of credit card data requires strict adherence to the Payment Card Industry Data Security Standard (PCI DSS) [1]. For many organizations, achieving and maintaining PCI compliance is a resource-intensive, expensive, and lengthy process. It requires rigorous audits, network segmentation, and constant security monitoring.

1.3 The PCI Vault Solution

PCI Vault mitigates these challenges by providing a secure, isolated environment for sensitive data storage. The core philosophy is to prevent sensitive data from ever traversing the Client's own infrastructure. By offloading the storage and transmission of sensitive data to PCI Vault, Clients can significantly reduce the scope of their own PCI compliance requirements while maintaining full control over their data through cryptographic proofs.

2 Infrastructure and Security Architecture

2.1 Serverless Environment

PCI Vault leverages a serverless cloud environment. This architectural choice offers several distinct security and operational advantages:

- **Reduced Attack Surface**: In a serverless model, the underlying operating system and runtime patching are managed by the cloud provider. This removes a significant burden on the security team and reduces the risk of unpatched server vulnerabilities.
- **Ephemeral Runtime**: Compute resources are ephemeral, spun up on-demand to handle requests, and destroyed immediately after. This lack of persistence makes it extremely difficult for attackers to establish a foothold or maintain presence within the infrastructure.
- **Isolation**: Functions run in isolated environments, ensuring strict separation between different processes and tenants.

2.2 Transport Layer Security (TLS)

All data in transit, whether via API requests or webhooks, is secured using Transport Layer Security (TLS). TLS provides privacy and data integrity between two communicating applications. It uses symmetric cryptography to encrypt data, ensuring that the connection is private, and asymmetric cryptography to authenticate the identity of the communicating parties. [2]

3 Key Management and Cryptography

3.1 Client-Controlled Security

A fundamental security tenet of PCI Vault is that while the system stores encrypted data, it does not persist the passphrases to the cryptographic keys required to decrypt it. Only the Client possesses this knowledge. The PCI Vault also allows for the rotation of these passphrases without affecting the ability to decrypt with the correct key derivation.

3.2 x25519 for Key Exchange and Encryption

PCI Vault utilizes **x25519**, a state-of-the-art elliptic curve Diffie-Hellman (ECDH) function, for its PGP-based key management. The formula for the curve is the following:

$$y^2 = x^3 + 486662x^2 + x$$

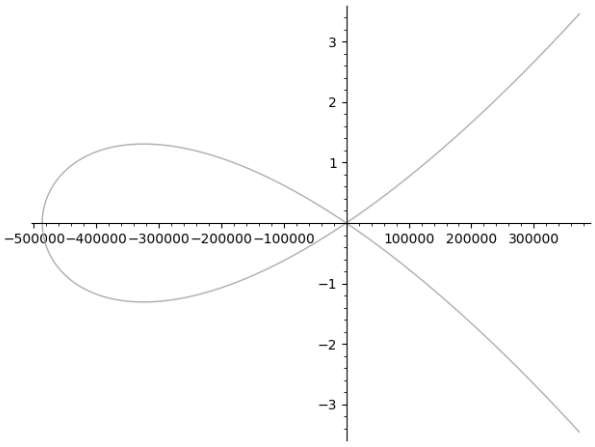


Figure 1: Visual representation of Curve25519

3.2.1 Benefits of x25519

- **Performance**: x25519 is designed for high performance. It uses Curve25519, which allows for extremely fast scalar multiplication compared to traditional algorithms like RSA or earlier elliptic curves (e.g., NIST P-256). This speed is crucial to maintain low latency in API responses.
- **Security**: Offers approximately 128 bits of security. Unlike some older curves, Curve25519 was constructed with a specific focus on avoiding implementation pitfalls. It is designed to be immune to timing attacks, where an attacker might infer the key based on how long an operation takes.
- **Key Size**: x25519 keys are only 32 bytes long. These small keys reduce storage requirements and bandwidth overhead during key exchange handshakes.

4 Data Vault and Tokenization Strategy

4.1 JSON Data Structure

PCI Vault stores data as JSON (JavaScript Object Notation) objects.

4.1.1 Benefits of JSON

- **Flexibility**: JSON is schema-less, allowing Clients to store arbitrary metadata alongside card numbers without database migrations.
- **Interoperability**: As the de facto standard for modern web APIs, JSON is natively supported by virtually all programming languages, simplifying integration for API Consumers.
- **Intelligent Parsing**: The vault intelligently parses nested JSON structures to locate and protect card or account numbers regardless of the document depth.

4.2 Tokenization Algorithms

Upon storage, sensitive numbers are replaced with tokens. These tokens act as safe handles to the data. PCI Vault employs specific algorithms for this process, relying heavily on the **BLAKE2s** hash function.

4.2.1 BLAKE2s Hashing

The tokenization engine uses the BLAKE2s cryptographic hash function.

- **Speed and Efficiency**: BLAKE2s is optimized for 8-bit to 32-bit platforms. It is faster than MD5, SHA-1, SHA-2, and even SHA-3, without compromising security. This efficiency allows PCI Vault to tokenize high volumes of data with minimal CPU overhead. [3]

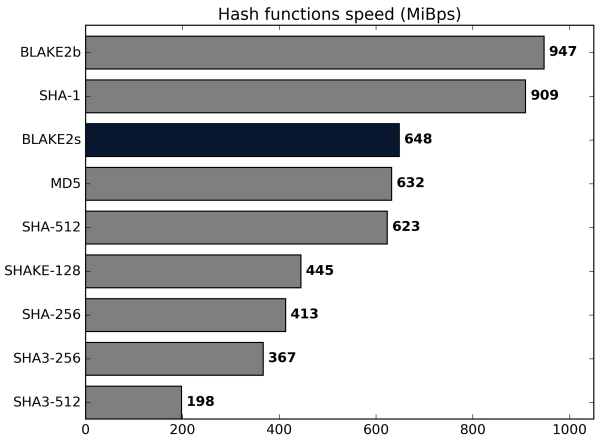


Figure 2: Blake2 speed (MiBps) vs other hashing algorithms

- **Security**: Offers a security margin similar to SHA-3 and includes features such as salt and personalization support, making it robust against length extension attacks.

4.2.2 Token Formats

1. **Full Tokenization**: The entire Primary Account Number (PAN) is replaced by a BLAKE2s hash representation. This provides maximum obfuscation.
2. **Partial Tokenization**: Preserves specific segments of the PAN (e.g., the first 6 and last 4 digits) while hashing the middle segment. This is critical for analytics and customer support use cases where identification is needed without revealing the full number.
3. **Luhn Algorithm**: The system supports generating tokens that pass the **Luhn check**.

- *Benefit*: The Luhn algorithm is a simple checksum formula used to validate a variety of identification numbers. By generating tokens that satisfy this checksum, PCI Vault ensures that systems performing basic validity checks on the token (e.g. legacy payment gateways) will not reject the token format, ensuring backward compatibility.

5 Secure Data Integration

5.1 Third-Party Forwarding

PCI Vault allows encrypted data to be forwarded directly to third parties (e.g., payment gateways) without the API Consumer ever seeing the plaintext data. This utilizes the secure infrastructure of the vault to decrypt and transmit data over a TLS-secured channel to the destination, ensuring the Client's infrastructure remains out of scope for PCI compliance.

5.2 Capture Endpoints and Webhooks

Unique capture endpoints enable external entities to insert sensitive data directly into the vault. This is paired with a webhook system that notifies the Client of successful data ingestion, ensuring synchronization between the vault and the Client's business logic.

5.3 Hosted HTML Forms with SRI

To facilitate direct end-user data entry, PCI Vault provides hosted HTML forms served via iFrames.

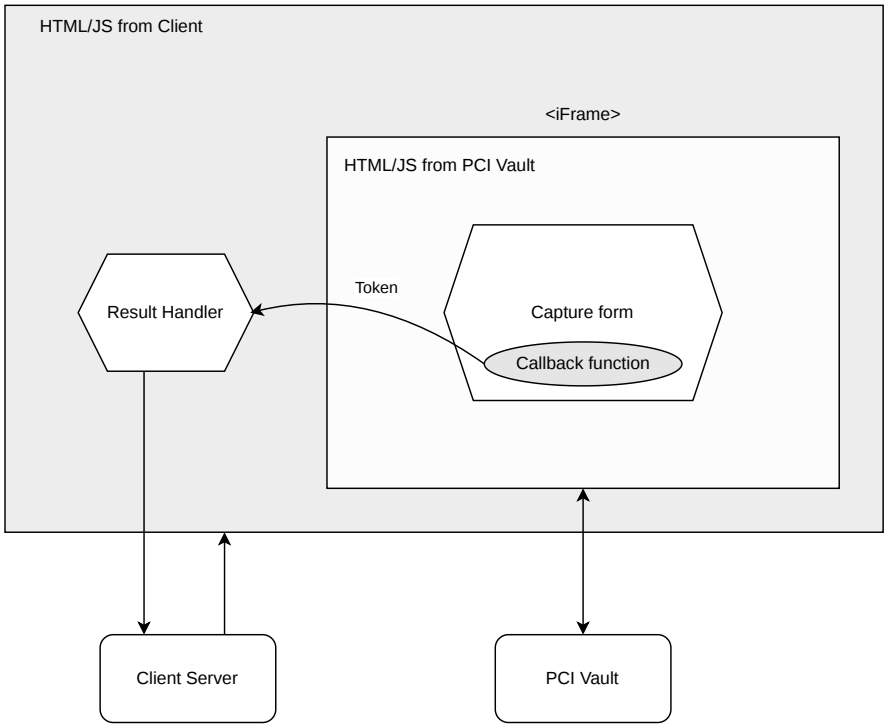


Figure 3: Embedded iFrame isolating sensitive data capture

5.3.1 Subresource Integrity (SRI)

Resources loaded within these forms, such as JavaScript and CSS, are protected using **Subresource Integrity (SRI)**.

Example of generating a SRI hash:

```
openssl dgst -sha384 -binary pci_vault_immutable.js | openssl base64 -A
```

- **Integrity Verification**: SRI enables the browser to verify that a fetched resource (e.g., from a CDN) has not been manipulated. It does this by checking the file's cryptographic hash against a provided hash in the HTML. [4]
- **Tamper Protection**: This ensures that even if the content delivery network is compromised, malicious code cannot be injected into the payment form, protecting the End User from cross-site scripting (XSS) attacks.

6 Rule Engine

A sophisticated rule engine allows Clients to define processing pipelines. This feature enables data transformation before storage or transmission. For example, data can be reformatted from a custom internal JSON structure to a specific format required by a third-party payment processor, all within the secure boundary of the vault.

Advanced rules such as hash, encrypt, and nonce are also supported. This allows complete control over proxy requests with zero-knowledge of the sensitive data stored.

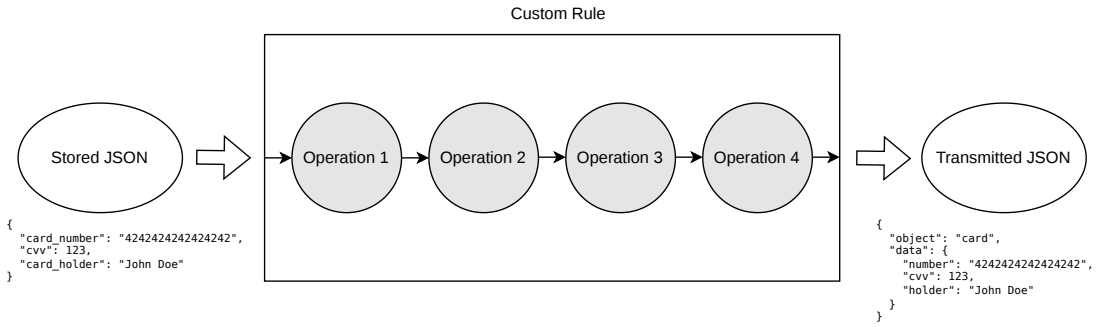


Figure 4: Advanced rules to process data

7 Conclusion

PCI Vault combines modern serverless infrastructure with robust cryptographic primitives such as x25519 and BLAKE2s to provide a secure, scalable, and compliant solution for payment data.

By leveraging standard formats like JSON and security mechanisms like SRI and TLS, it ensures interoperability and end-to-end security, effectively solving the burden of PCI compliance for its Clients.

References

[1] "Payment Card Industry Data Security Standard". *W*ikipedia, Retrieved March 23, 2026.

[2] "SSL/TLS in Detail". *T*echNet. Microsoft Docs. Retrieved October 24, 2010.

[3] "BLAKE2 — fast secure hashing". *b*lake2.net, Retrieved on 3 April, 2016.

[4] "Subresource Integrity". *M*DN, Retrieved January 3, 2025.